Quando la tua applicazione Django non va abbastanza veloce

Riccardo Magliocchetti

whoami

Software developer @ Maieutical Labs

Consultant

OSS contributor

Ridiculously fast

Django was designed to help developers take applications from concept to completion as quickly as possible.

www.djangoproject.com

La mia esperienza

Ottimizzazioni di piccoli e medi progetti Django in piccoli team

DUE REGOLE PER OTTIMIZZARE

1. Don’t do it

1. Don't do it.
2. (for experts only)
   Don’t do it yet.

1. **Don't do it.**
2. **(for experts only) Don't do it yet.**
   Not until you have a perfectly clear and unoptimized solution.

Michael A. Jackson
Principles of Program Design, 1975

Hai raggiunto
il product/market fit?

Quando pensare alle performance

***Idealmente* prima di andare in produzione**

Code review: *Quante query fa questo endpoint?*

questa URL è lenta

Clienti insoddisfatti

uso *troppa* RAM

Risorse sprecate

COME SI OTTIMIZZA?

Fare meno lavoro

Il codice sta facendo qualcosa che non serve

Far meglio

Migliore algoritmo o struttura dati

📚 **High Performance Python, *Ian Ozsvald, Micha Gorelick***

MISURA PRIMA DI OTTIMIZZARE

Monitoring

SaaS: Datadog, New Relic, Elastic APM, etc...

Prometheus, talk Pycon9 @davidesetti

TOOL OPEN SOURCE

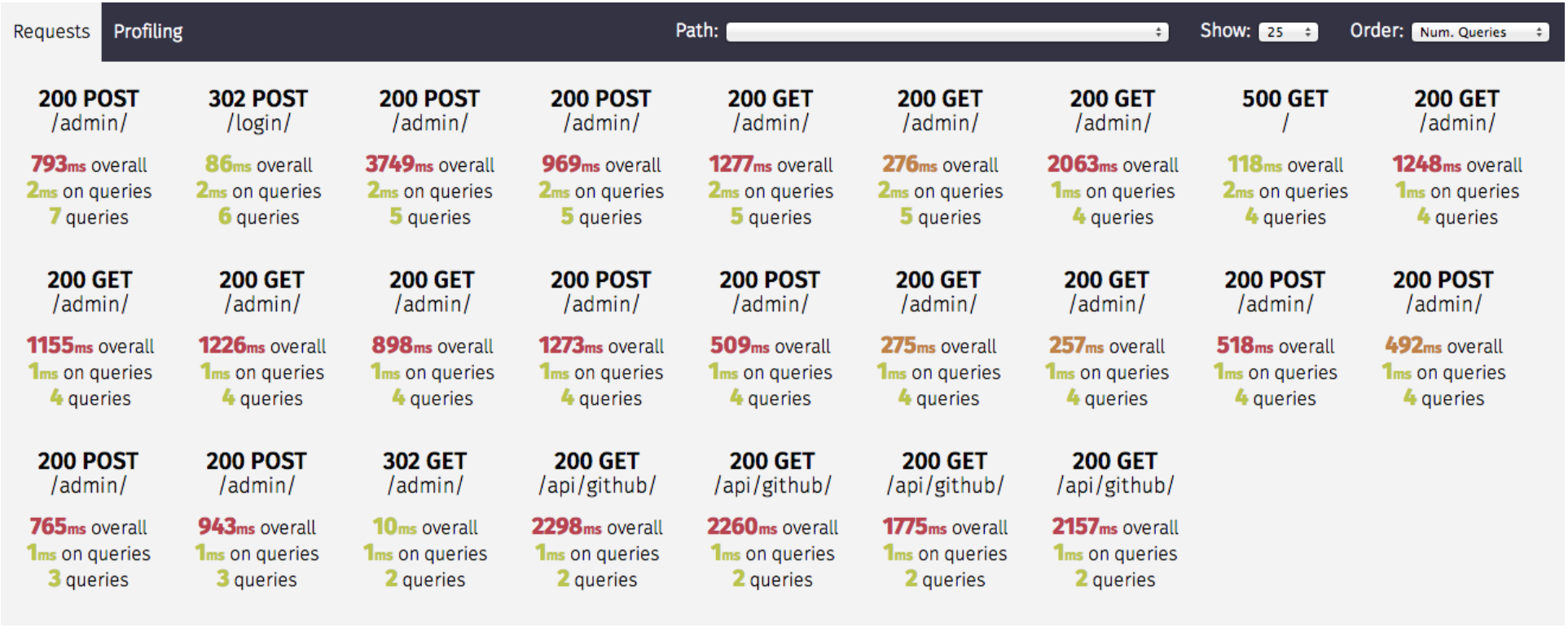
MISURARE RICHIESTE LENTE

jazzband/django-silk

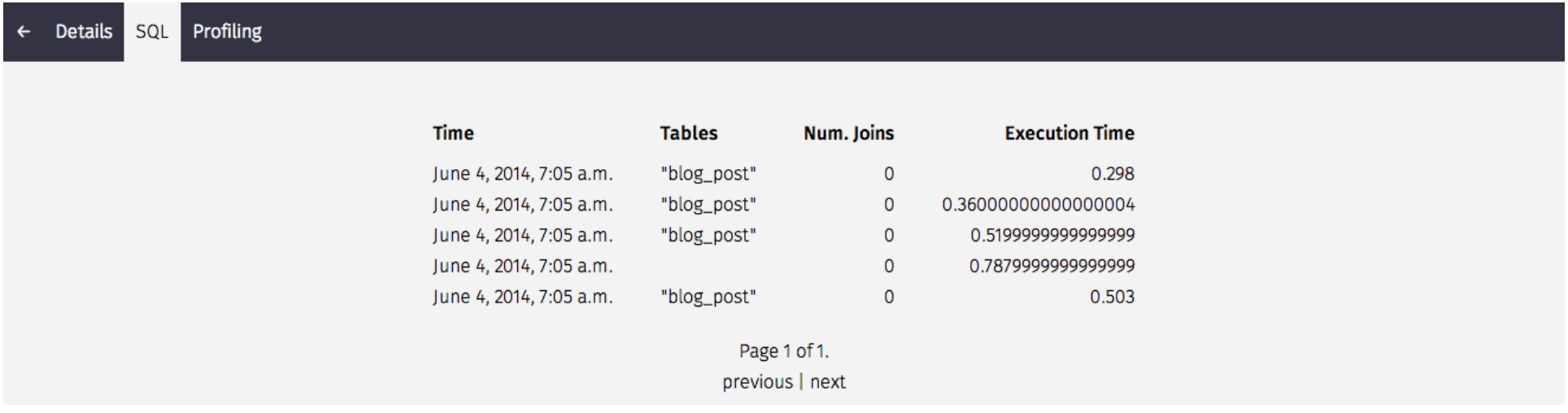
Profilazione e introspezione di richieste

Online, non in produzione

Dashboard richieste



Query per richiesta



SQL query

← SQL Query

```
SELECT "blog_post"."id",
       "blog_post"."published_date",
       "blog_post"."modified_date",
       "blog_post"."revision",
       "blog_post"."markdown",
       "blog_post"."published"
FROM "blog_post"
WHERE "blog_post"."revision" IS NOT NULL
ORDER BY "blog_post"."published_date" DESC
```

0.562ms

0 joins

Traceback

The below is the Python stacktrace that leads up the execution of the above SQL query. Use it to figure out where and why this SQL query is being executed and whether or not it's actually neccessary.

File "/Users/mtford/Playground/website_blog/silk/sql.py", line 30, in execute_sql
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/db/models/sql/compiler.py", line 713, in results_iter

```
...
        return columns

    def results_iter(self):
        """
        Returns an iterator over the results from executing this query.
        """
        resolve_columns = hasattr(self, 'resolve_columns')
        fields = None
        has_aggregate_select = bool(self.query.aggregate_select)
        for rows in self.execute_sql(MULTI):
            for row in rows:
                if has_aggregate_select:
                    loaded_fields = self.query.get_loaded_field_names().get(self.query.model, set()) or self.query.select
                    aggregate_start = len(self.query.extra_select) + len(loaded_fields)
                    aggregate_end = aggregate_start + len(self.query.aggregate_select)
                if resolve_columns:
                    if fields is None:
                        # We only set this up here because
                        # related_select_cols isn't populated until
                        # execute_sql() has been called.

...
```

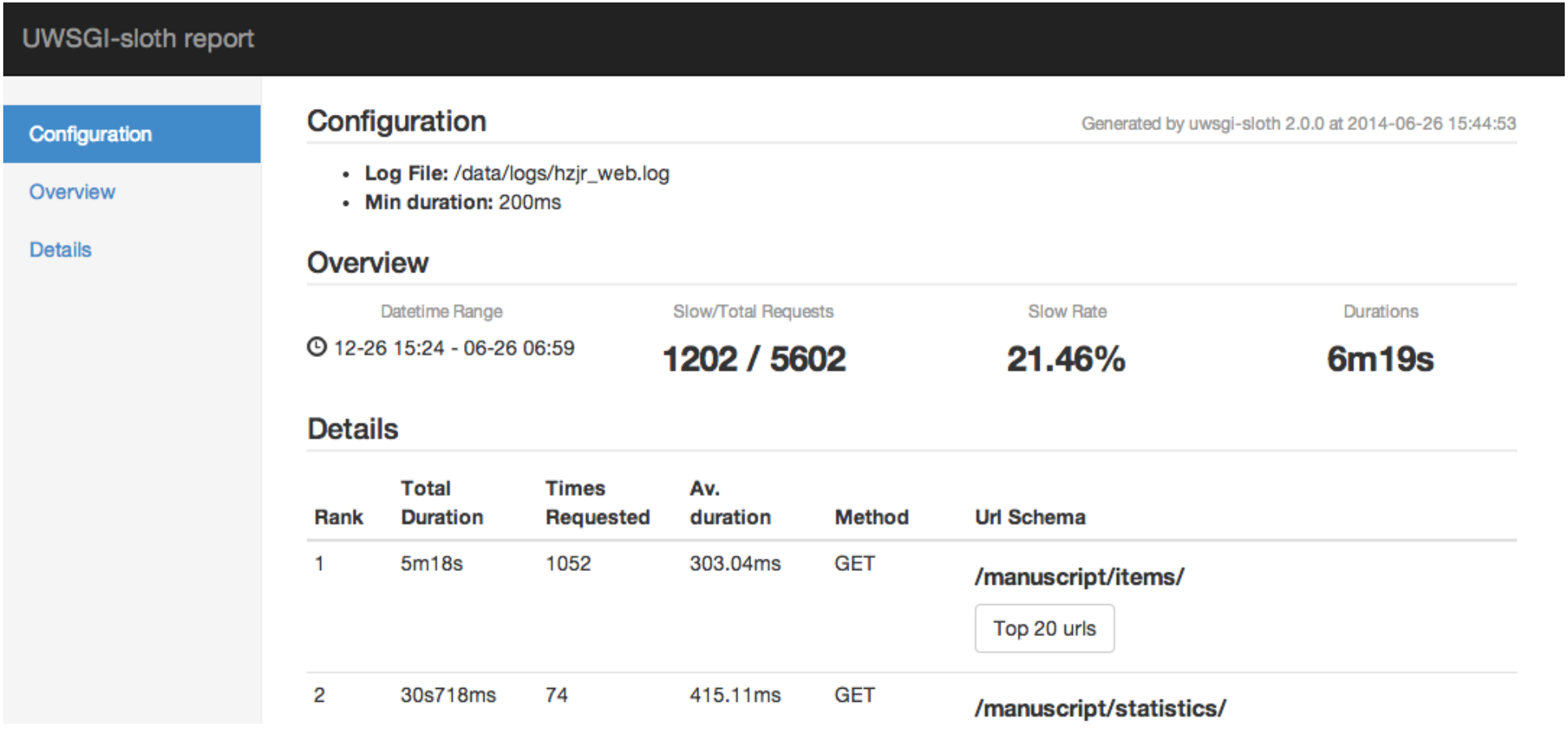
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/db/models/query.py", line 220, in iterator
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/db/models/query.py", line 857, in _fetch_all
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/db/models/query.py", line 77, in __len__
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/template/defaulttags.py", line 156, in render
File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/django/template/debug.py", line 78, in render_node

piglei/uwsgi-sloth

Analizza richieste lente dai log

Offline, per uWSGI

Attenzione alla media!


UWSGI-sloth report
Configuration
Overview
Details
Configuration
Generated by uwsgi-sloth 2.0.0 at 2014-06-26 15:44:53
Log File: /data/logs/hzjr_web.log
Min duration: 200ms
Overview
Datetime Range
12-26 15:24 - 06-26 06:59
Slow/Total Requests
1202 / 5602
Slow Rate
21.46%
Durations
6m19s
Details
Rank
Total Duration
Times Requested
Av. duration
Method
Url Schema
1
5m18s
1052
303.04ms
GET
/manuscript/items/
Top 20 urls
2
30s718ms
74
415.11ms
GET
/manuscript/statistics/

MISURARE CONSUMO MEMORIA

xrmx/pyuwsgimemhog

Trova viste con possibili memory leak dai log

Offline, per uWSGI

Come ottimizzare in pratica

1. Fare un database fatto bene
2. Fare meno query
3. Fare query meno costose
4. Fare meno lavoro inutile

Un database

```
User
  <- Profile
    <- Specific profile
      <- Specific Django reusable app profile
```

Un database più veloce

```
Custom User
  <- Specific profile
  <- Specific Django reusable app profile
```

L’ereditarietà concreta dei modelli Django è una trappola

Usa quella astratta o esplicitamente una relazione

QUERY

Prendere solo quello che serve

`values()` e `values_list()`

Salvare solo quello che serve

```
-     score.save()
+     score.save(update_fields=['points'])
```

Le query N + 1 uccidono le performance dell'applicazione

```
class Room:
    building = models.ForeignKey(Building)

class Appointment:
    room = models.ForeignKey(Room)
    doctor = models.ForeignKey(Doctor)
```

```
a = Appointment.objects.get(
    id=appointment_id
)
print('In {} with {}'.format(a.room, a.doctor))
```

Per evitare query N + 1

- `select_related()`, JOIN in SQL
- `prefetch_related()`, join in Python

select_related()

```
class Room:
    building = models.ForeignKey(Building)

class Appointment:
    room = models.ForeignKey(Room)
    doctor = models.ForeignKey(Doctor)
```

```
-   a = Appointment.objects.get(
+   a = Appointment.objects.select_related(
+       'doctor', 'room__building'
    ).get(
        id=appointment_id
    )
```

select_related() è greedy

```
+  ).select_related(
+      'doctor', 'room__building'
   )                ^^^^^^^^^^^^^^^^
```

SELECT * in JOIN: doctor, room e building

Attenzione a select_related()

- L'admin se nel `list_display` c'è un solo campo preso da una relazione seleziona tutti i modelli delle relazioni
- Memory leak in django 1.11?

Attenzione alle librerie

```
children = ListField(
      child=RecursiveField(),
-     source='children.all'
+     source='get_children'
)
```

Attenzione alle librerie

```
Do less work in MySerializer

Use django-mptt get_children method instead of
going straight to the field model.

/myapi/ does 40% less queries
```

Il numero di query è testabile

`django.test.TransactionTestCase.assertNumQueries()`

```
with self.assertNumQueries(2):
    Person.objects.create(name="Aaron")
    Person.objects.create(name="Daniel")
```

INDICI

Indici: se filtriamo o ordiniamo

```
-     complete = models.BooleanField(default=False)
+     complete = models.BooleanField(
+         default=False,
+         db_index=True
+     )
```

Ordinare non è gratis

```
class MyAdmin:
     readonly_fields = ['creation_ts']
-    ordering = ['creation_ts',]
```

```
Remove unuseful ordering in MyAdmin

Ordering a timestamp without indexes is slooooooow.
Now the query takes 30 seconds less!
```

Gli indici non sono gratis

```
class MyModel:
    elements = models.ManyToManyField(Element)
```

Indici nella tabella di join:

1. PK
2. FK verso MyModel
3. FK verso Element
4. UNIQUE sulle FK

Gli indici ci servono sempre?

```
-     elements = models.ManyToManyField(Element)
+     # Be aware we don't have indexes here
+     elements = ArrayField(
+         models.IntegerField(),
+         default=[],
+         blank=True
+     )
```

Se gli indici non ci servono

```
Use an array field for MyContainer elements

Instead of using an m2m. We don't need the field
for the application, we need it for analytics.

This let us save 2GB from tables and indexes.
```

SERIALIZZATORI

Usare il serializzatore giusto

```
class MySerializer(serializers.ModelSerializer):
-     owner = UserSerializer()
+     owner = serializers.IntegerField(
+         read_only=True
+     )
```

Esplicito meglio di implicito #1

```
class MySerializer(serializers.ModelSerializer):
    class Meta:
        model = MyModel
        fields = '__all__'
```

Esplicito meglio di implicito #2

```
class MySerializer(serializers.ModelSerializer):
    class Meta:
        model = MyModel
        exclude = ('owner',)
```

Non usare i ModelSerializer per serializzare

Il context ti è amico

```
class MySerializer:
-    title = serializers.CharField(
-          source='assignment.task.title'
-    )
+    title = serializers.SerializerMethodField()

+    def get_title(self, obj):
+        titles = self.context.get('titles')
+        return titles[obj.assignment_id]
```

`select_related()` e **`prefetch_related()`**
sarebbero state dispendiose

Altre cose da guardare

- `defer()` e `only()`
- `union`
- `Prefetch`
- `aggregate()` e `annotate()`
- *Database functions* e la documentazione del Database
- `FilteredRelation()` django ≥ 2
- `EXPLAIN`

Letture interessanti

- The Dramatic Benefits of Django Subqueries and Annotations
- Django ORM Cookbook
- Updating a Django queryset with annotation and subquery

Conclusioni

- **Misura prima di *eventualmente* ottimizzare**
- **Pensa alle performance**
- **Piccoli cambiamenti possono dare grandi risultati**
- **Conoscere SQL aiuta**

Grazie! Domande?

github.com/xrmx

speakerdeck.com/xrmx

@rmistaken